CMPSCI 240
Reasoning Under Uncertainty
Homework 6

Prof. Hanna Wallach

Assigned: April 6, 2012
Due: April 13, 2012

Overview

For this project, you will build a Naive Bayes classifier which will classify text strings into one of two possible categories or *classes*. Your program will learn the difference (as best as it can) between the two classes by examining a collection of pre-classified example strings. Then, your program will see how well it can generalize what it has learned to new strings it has not seen before. The features your classifier will use are the presence or absence of each of the letters of the alphabet. Multiple occurrences of a letter in a string should not affect your classifier—only whether the letter is present or not.

How the Program Works

The program operates in two phases. First, in the *training phase*, your program reads in two different text files containing training examples for the two classes. These two files together are called the *training set.* Your program will examine the strings in the training set and, for each class, tabulate information about the number of training examples and how many times each letter of the alphabet appears in the examples for that class.

Second, in the *testing phase*, your program will read in another two text files containing a new set of examples for each of the two classes. These two files together are called the *testing set.* Your program will run the classifier on each string in the testing set and classify it into one of the two classes by identifying the MAP hypothesis. You will also compute the posterior probability of each example belonging to each of the two classes. (There is code included that takes care of formatting the output for doing this nicely and evaluating how well your classifier performed by counting the number of examples in the testing set that were indeed classified correctly.)

It is not realistic to think that your classifier will perform perfectly. After all, the only information your classifier will have to work with is which of the 26 letters are present in any new string it is trying to classify. Therefore, do not be concerned if the program reports a classifier accuracy below 100%, that does not necessarily imply your program is doing something wrong.

The Skeleton Code

You can run the skeleton Java file `TrainAndTest.java` immediately. It will ask you for the two text files for the training set and two text files for the testing set. The first file given for the training and testing sets should contain examples from one class (called class 0) and the second file should contain examples from the other class (called class 1). The names of the classes are 0 and 1 because the classifier doesn't know what sorts of strings are in the files. Note that `TrainAndTest.java` won't do anything intelligent right out of the box, but it should compile and run just fine.

For the training phase, you will need to fill in these methods:

- `NaiveBayesClassifier.addTrainingExample(...)`: This method is called once for each string in the training set.
- `Feature.updateFeatureCounts(...)`: This method is called by `NaiveBayesClassifier.addTrainingExample(...)` to update the frequency tables of how often this feature appears in examples.

For the training phase, you will need to fill in these methods:

- `NaiveBayesClassifier.getPriorProbability(...)`: This method returns the prior probability for a class.
- `NaiveBayesClassifier.getLikelihood(...)`: This method returns the likelihood for a class and string.
- `NaiveBayesClassifier.getPosteriorProbability(...)`: This method returns the posterior probability for a class and a string.
- `NaiveBayesClassifier.classify(...)`: This method returns the maximum a posteriori (MAP) hypothesis for a string.

You are given some example data files to play around with: lists of US cities, Russian cities, and other cities, along with a collection of Tweets on various different topics. You can see how well your classifier does, for example, in trying to distinguish the name of a US city from a Russian one, or figuring out if a Tweet originated from a Republican or a Democrat (these data come from the 2008 election season), or Mitt Romney or Rick Santorum (these data come from 2012 Tweets by those two candidates).

Turning in the Program

You should upload your program to your account on the EdLab: create a subdirectory called `hw6` inside your `cs240` directory and upload your `Feature.java` and `NaiveBayesClassifier.java` files into it.